VIPER

VOLUME 2, ISSUE 6 AN ARESCO PUBLICATION JANUARY 1980, $2.00

TABLE OF CONTENTS

Please Notice the addition of a complete new section to our newsletter: PIPS MODS. This is in response to the great number of readers who have offered hints, suggestions, and mods (as Tom himself requested) - and who have taken the time to write them us for all of us!

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

(Required by 39 U.S.C. 3685)

1. TITLE OF PUBLICATION	A. PUBLICATION NO.	2. DATE OF FILING
VIBES	PENDING	11/1/79

3. FREQUENCY OF ISSUE	A. NO. OF ISSUES PUBLISHED ANNUALLY	B. ANNUAL SUBSCRIPTION PRICE
MONTHLY EXC. JUNE + DEC	10	15.00

4. LOCATION OF KNOWN OFFICE OF PUBLICATION *(Street, City, County, State and ZIP Code) (Not printers)*
6303 GOLDEN HOOK COLUMBIA MD 21044 (HOWARD CO.)

5. LOCATION OF THE HEADQUARTERS OR GENERAL BUSINESS OFFICES OF THE PUBLISHERS *(Not printers)*
Same

6. NAMES AND COMPLETE ADDRESSES OF PUBLISHER, EDITOR, AND MANAGING EDITOR

PUBLISHER *(Name and Address)*
ARESCO INC 6303 GOLDEN HOOK Columbia MD 21044

EDITOR *(Name and Address)*
Terry L Lauderman Same

MANAGING EDITOR *(Name and Address)*
Terry L Lauderman Same

7. OWNER *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be giving. If the publication is published by a nonprofit organization, its name and address must be stated.)*

NAME	ADDRESS
Terry L Lauderman	6303 Golden Hook Columbia MD 21044
RICHARD SIMPSON	Same

8. KNOWN BONDHOLDERS, MORTGAGEES, AND OTHER SECURITY HOLDERS OWNING OR HOLDING 1 PERCENT OR MORE OF TOTAL AMOUNT OF BONDS, MORTGAGES OR OTHER SECURITIES *(If there are none, so state)*

NAME	ADDRESS
NONE	

10. EXTENT AND NATURE OF CIRCULATION	AVERAGE NO. COPIES EACH ISSUE DURING PRECEDING 12 MONTHS	ACTUAL NO. COPIES OF SINGLE ISSUE PUBLISHED NEAREST TO FILING DATE
A. TOTAL NO. COPIES PRINTED *(Net Press Run)*	1000	1000
B. PAID CIRCULATION 1. SALES THROUGH DEALERS AND CARRIERS, STREET VENDORS AND COUNTER SALES	203	203
2. MAIL SUBSCRIPTIONS	484	484
C. TOTAL PAID CIRCULATION *(Sum of 10B1 and 10B2)*	687	687
D. FREE DISTRIBUTION BY MAIL, CARRIER OR OTHER MEANS SAMPLES, COMPLIMENTARY, AND OTHER FREE COPIES	10	10
E. TOTAL DISTRIBUTION *(Sum of C and D)*	697	697
F. COPIES NOT DISTRIBUTED 1. OFFICE USE, LEFT OVER, UNACCOUNTED, SPOILED AFTER PRINTING	303	303
2. RETURNS FROM NEWS AGENTS	0	0
G. TOTAL *(Sum of E, F1 and 2—should equal net press run shown in A)*	1000	1000

11. I certify that the statements made by me above are correct and complete.

SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER
Terry L. Lauderman Editor

12. FOR COMPLETION BY PUBLISHERS MAILING AT THE REGULAR RATES *(Section 132.121, Postal Service Manual)*

39 U. S. C. 3626 provides in pertinent part: "No person who would have been entitled to mail matter under former section 4359 of this title shall mail such matter at the rates provided under this subsection unless he files annually with the Postal Service a written request for permission to mail matter at such rates."

In accordance with the provisions of this statute, I hereby request permission to mail the publication named in Item 1 at the phased postage rates presently authorized by 39 U. S. C. 3626.

SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER
Terry L. Lauderman Editor

Terry - This is a plea for help. Does any Viper reader know of a BASIC interface or compiler (that would be nice) which can be stuffed into 6-8K (ROM, EPROM, or RAM) and which is fast, has a string function, exponential function, some kind of graphics, named cassette files, FOR/UNTIL/NEXT and can handle numbers greater than 32K, and can run on the Netronics ELF II?

While Tiny BASIC is fantastic (and Tom Pittman is a genius), it is a bit too slow...and my children can't write machine code subroutines at the moment. So it is a bit limited (no pun intended) for teaching the finer points of BASIC.
-John McLaren

Terry- About my comments on PIPS for VIPS - I found out why the letters of the character set of the Space Wars program were garbage while the numbers were O.K. As per the instructions included with the cassette, I loaded 8 pages. This loaded only the first half of the ASCII set, which runs from 0700 to 0AFF, as I found by studying the manual. When I load B pages the messages are all there. Other comments apply.
-John B. Sewell

Terry - I have recently pruchased some 6514 CMOS RAMS from Emerge Systems (Ref: VIPER 2.02.28) for use on my COSMAC VIP. Although 6514's are pin compatable with 2114, there is a memory address timing problem during a read cycle. This is due to the address latch in the 6514. I made the following mod to my VIP to allow the use of either 2114's or 6514's.

1) Jumper U4 pin 3 to U4 pin 11, U5 pin 10 to U4 pin 13, and U4 pin 8 to U9 pin 13;
2) Cut the etch between U9 pins 13 and 14 on the top of the board. (I was able to cut the etch without removing the socket but since the spacing is so close, socket removal may be required.) This mod delays the memory chip enable during a read cycle until the lesser significant address bits are on the bus and may be latched by the 6514's. The memories work very well and reduce my power requirements enough that I don't need to change power supplies.

-Steven Wilcox

Terry - I don't know if it will be big news to VIPER readers, but I filled up the empty RAM scokets in my VIP I with some low-power 2114's from EMM semi. I measured the current into the VIP without the RAMs at 300mA. With the EMM Semi low-power RAMS in, the current rose to 500mA -- with the beeper on. That's well within what the RCA-supplied power supply is supposedly good for.

According to Roy Evans at EMM Semi, CMOS RAMs take less current than even his do but he said that you can't get the CMOS ones anywhere. -Chuck Small

Rick - I need help. I can program adequately in BASIC, somewhat in Chip-8 and a little in 6800 machine language but I cannot grasp 1802 machine language programming. The RCA manual (MPM-201B) is not much help; I have the feeling that it is telling me how to run a chicken ranch when what I want is directions on how to boil an egg.

I would like to see in the VIPER a continuing column, or a few long articles, on 1802 machine language. If the project is not too ambitious, a PIPS type book with maybe a computer aided instruction tape might be a better approach. Am I alone in my ignorance? -John Sewell

---

I am currently a disappointed owner of the VP550 Super Sound Board. I'm disappointed because I cannot get it to work.

I've checked and rechecked the pin-8 interpreter code and sample tune that is provided in the manual, and I have entered it as printed.

I'm confused by the documentation, which claims in Appendix III that the Super Sound system is controlled by writing parameters to memory locations in the 8000 range. I am under the impression that the ROM OS occupies this memory range, and that there is no way to modify this other than removing the ROM chip.

I'm hoping that someone else has a Super Sound system up and running and can give me a hand. I suspect that the pin-8 interpreter listing in the dicumentation has a few bugs or typos, and that various other parts of the dicumentation are simply incorrect. I would appreciate it if anyone who can help would write to me at this address: Bob Kantor, 144 Baltic Street, Brooklyn, NY 11201.*

*Replies to Aresco so that they may be printed are welcome.

Terry - I believe sending in my subscription renewal the day before yesterday had some positive side effects!

I don't know from beans about computers - in fact, I've been spelling it wrong for some time. I'm just barely learning what makes my ELF II tick. I heard about the ELF to VIP conversion and did it. After correcting a few of my own key errors I got it running. One problem : my 4's kept coming up with a base + 1 dot 4.

This makes for a nice stable 4 - it can't fall over but the extra sure screws up a following . It prints .

I believe the conversion software is in error. MEM (orig) 8104 changed to 0E04 should be 3E not 3F. All works fine now. -Walt Pinner

HI-RES CHIP-8

by Tom Swan

HI-RES means "high resolution" and does not refer to any brands of soft drinks. Any reference to real sode pop, sparkling or flat, is pure coincidence. (Editor's note)

Some months ago the VIPER presented a two page, medium resolution modification for Chip-8. The following modifications extend the VIP display software as far as possible without hardware changes. You will now be able to program high resolution graphics without needing to learn 1802 machine language.

One unexpected benefit from these modifications was an increase of speed. I hesitated writing this program for the longest time assuming the four page display software to run slower than a one page setup. I did not have me thinking cap on! The interrupt routine for a four page display is the simplest affair, only a quick initialization for the DMA pointer R0 and that's about it. No loops!

Comparing the two interrupt routines, and simply counting the number of instructions executed during each cycle, tells us just how much faster we can go. The interrupts occur every 1/60 seconds and this is true regardless of the display format. But the one page interrupt routine executes 386 additional instructions than the four page variety which has no time killer loops. That's 23,160 instruction executions per second less for four pages than for one! Holy Concorde Batman!

(Note: I am taking DMA timing and Chip-8 interrupt features such as the timer as constants here -- the actions are the same for all interrupt formats. The figure above comes from multiplying the number of instructions (12) of the inner interrupt loop x 32 loops. This plus the two E2 NOPs just before the loop equals 386 instructions more than the loopless four pager.)

A WORD ON MODIFICATIONS

Some of the comments I have received from readers complain of a lack of standardization among the many modifications to Chip-8 presented in VIPER. To me one of the advantages to having an interpreter in RAM is that it can be modified but I agree that thirty different Chip-8 languages may actually be counter productive to programming unless you keep good notes on the changes.

To offer a final, standard Chip-8 including all modifications, however, would be to assume no further changes to those modifications will be invented. The VIP is a young computer, its growth fed by VIPER and all of you inventive programmers who own one. Maybe someday the ideal Chip-8 will be offered, but not yet.

I suggest that any modifications you do make be kept separate on tape. Do not destroy your original versions of Chip-8 when adding new features. Instead, keep copies of all versions. With so many variations becoming possible, I am starting a Chip-8 tape -- one cassette dedicated to storing all the many types of Chip-8 languages currently available. You may wish to do the same.

THE PROGRAM

Begin with a copy of your normal one page resolution Chip-8, the version presented in your VIP manual. Make all of the changes here being careful to note the proper addresses for each modification. At 00DB you need to enter the proper hex value as noted depending on the amount of memory you have. When you are done, record three pages from 0000-02FF and mark the program HI-RES CHIP-8.

The following conditions reflect the changes you have made:

1) 0200 = ERASE DISPLAY -- do not use the old 00E0 erase instruction any more. It has been deactivated and its use is a sure invitation to a program failure. All other Chip-8 instructions are valid.

2) Chip-8 variables, work space, and stack are configured as before according to the Chip-8 memory map in your manual. The location of these items is now on the memory page (0Z=0X-4) where X = the highest RAM page in your system. (4K/0Z=0B; 3K/0Z=07; 2K/0Z=04)

3) Screen resolution is 64 bits horizontal (Vx) and 128 vertical (Vy). This corresponds with the hex values 3F and 7F.

4) An additional bonus instruction has been added:

   0216 = PROTECT -- V0 = # pages

   Using this instruction will cause an internal change to be made to the erase display instruction, 0200. Set Vo to the number of memory pages you want to clear on the next and subsequent uses of the 0200 erase display command. Then call, 0216 PROTECT, a machine language subroutine. Vo must not be set to any values other than 1,2,3,4. The following programming example demonstrates the use of the 0216 command. It will cause only the bottom half -- two of the four display memory pages -- to be cleared.

```
6002;Vo=02--Set Vo to # pages for clearing
0216;PROTECT--Change display instruction
0200;ERASE--Erase 2 pages of display
```

This instruction may be used to protect titles or scores, etc. in the top of the display, while erasing the rest.

5) Chip-8 programs now begin at 0244 and may continue to 0Z9F where Z = 0X-4 as explained in #2. This is enough room for the demonstration programs in this issue even if you have the smallest 2K VIP setup.

6) The interpreter must not extend beyond 0243 or make use of memory locations 01F2-01F9. These areas will be needed in order to include the Messager capability from PIPS-I. Details next month. (If you do not have a copy of PIPS-I you may disregard this restriction.)

HI-RES CHIP-8

```
0003  04          ;Cause R6.1 and R2.1 to = 0Z = 0X-4
----
000A  02          ;High part new interrupt routine address
----
000D  25          ;Low part   "      "        "       "
----
0018  FA          ;New start first Chip-8 instructions @ 01FA
----
004E  C0 LBR ERASE;Long branch to erase display following
  4F  02                video on command. May cause slight
  50  00                and unobjectionable flicker once on run)
----
007E  7F          ;New lower Vy limit = 127 decimal
  7F  E2 SEX 2    ;NOP--remove shift instruction
----
0082  30 BR       ;Branch to patch at 00E0
  83  E0
  84  74 ADC      ;Finish patched multiply to set
  85  BC PHI RC   ;    RC = destination address for display
----
00B5  FC          ;Branch point to 00FC (change from 00D9)
----
00C5  D1          ;Branch point to 00D1 (change from 00D2)
----
00D0  73 STXD     ;Combines the previous 5C/2C instruction pair
  D1  16 INC R6   ;Increment display bits pointer
  D2  8C GLO RC   ;Add 8 to display address to point to
  D3  FC ADI      ;    a byte just under the last one
  D4  08          ;    displayed
  D5  AC PLO RC   ;Put this value in RC.0
  D6  9C GHI RC   ;Add possible carry if page boundary
  D7  7C ADCI     ;    was crossed to RC.1
  D8  00
  D9  BC PHI RC   ;Put this value in RC.1
```

```
00DA  FB  XRI      ;Compare RC.1 value with overflow page
  DB *08/0C/10     ;*Enter - 2K/08; 3K/0C; 4K/10
  DC  3A  BNZ      ;If comparison fails, continue with
  DD  B3           ;    display routine
  DE  30  BR       ;Otherwise branch to exit -- display
  DF  FC           ;    has reached the bottom

00E0  AC  PLO  RC ;Value in D was shifted left twice earlier
  E1  93  GHI  R3 ;R3=00 always -- this saves space only
  E2  7E  SHLC    ;Shift in possible carry from previous SHL @ 0081
  E3  BC  PHI  RC ;Store for the moment in RC.1
  E4  8C  GLO  RC ;Get RC.0 value (=(Vy MOD 7F_h)x 4)
  E5  FE  SHL     ;Shift another time left for^h (Vy x 8)
  E6  F1  OR      ;"OR" in Vx byte address on stack
  E7  AC  PLO  RC ;And put completed value in RC.0
  E8  9C  GHI  RC ;Repeat with possible carry to high
  E9  7E  SHLC    ;    byte in RC.1
  EA  52  STR  R2 ;Push this value onto stack
  EB  9B  GHI  RB ;Get display page address
  EC  30  BR      ;Return from patch, at 0084, program
  ED  84          ;    will add RB.1 + RC.1 to complete
                  ;    the creation of an address in RC
                  ;    corresponding to the 64 x 128
                  ;    display matrix
----
00FC  F8  LDI      ;Set R6 to address Chip-8 VF variable
  FD  FF           ;    which is the "hit indicator"
  FE  A6  PLO  R6  ;    needed by some games
  FF  87  GLO  R7 ;Get saved indicator value in R7.0

0100  56  STR  R6 ;Store as new VF value (=00 or 01)
  01  12  INC  R2 ;Reset stack pointer (decremented @ 007A)
  02  D4  SEP  R4 ;Return -- display complete

----                NOTE -- a page boundary was crossed at
                            0100.  While this does not
                            represent "pure programming
                            practices," no short branches
                            are involved and it is an
                            acceptable solution to a severe
                            space shortage.
```

FIRST CHIP-8 INSTRUCTIONS

```
01FA  023F;ADJUST -- Do MLS -- Adjust RB.1 to 4 page display
01FC  004B;TV-ON  -- Do MLS -- Turn TV on/auto page erase
01FE  1244;BEGIN  -- Goto start user program @ 0244

                    NOTE -- these instructions form a
                            permanent part of the interpreter.
                            If you prefer programs to begin at
                            0300 or other area, the jump command
                            at 01FE may be altered for your
                            own requirements.
```

*Be sure to enter the correct value here for your system.
4 memory chips = 2K
6 memory chips = 3K
8 memory chips = 4K (8K or larger systems use this value usually)

4-PAGE ERASE

```
0200  9B  GHI   RB ;RB.1 holds display page
  01  FC  ADI      ;Add 3 to address last page
  02  03           ;    of display
  03  BF  PHI   RF ;Put in RF.1 serving as a pointer
  04  F8  LDI      ;    for erasing from the bottom up
  05  FF
  06  AF  PLO   RF ;Set RF.0 = FF/RF = 0XFF here
  07  F8  LDI
  08  04           ;(this byte subject to change by "PROTECT")
  09  AE  PLO   RE ;Set up RE.0 = loop count of 4 ("N")
  0A  EF  SEX   RF ;X = F  facilitates speed of loop
  0B  94  GHI   R4 ;Get 00 byte from R4.1
  0C  73  STXD     ;Store @ M(R(F)) to erase one byte
  0D  8F  GLO   RF ;Test pointer
  0E  3A  BNZ      ;Loop to erase one page (note
  0F  0B           ;    that last byte is ignored)

0210  2E  DEC   RE ;Count pages erased (RE.0-1)
  11  8E  GLO   RE ;Test count in RE.0
  12  3A  BNZ      ;Loop to erase "N" pages less
  13  0B           ;    a single byte
  14  5F  STR   RF ;Erase last byte-(allows short inner loop)
  15  D4  SEP   R4 ;Return
```

PROTECT

```
0216  F8  LDI
  17  F0           ;Set R6 to address Chip-8 variable Vo
  18  A6  PLO   R6
  19  F8  LDI
  1A  02
  1B  BF  PHI   RF ;RF.1 = page address erase routine
  1C  F8  LDI
  1D  08
  1E  AF  PLO   RF ;RF addresses byte in erase routine
  1F  06  LDN   R6 ;Get value of Vo

0220  5F  STR   RF ;Store as new erase loop counter
  21  D4  SEP   R4 ;Return
```

4-PAGE INTERRUPT

```
0222  7A `REQ      ;Q = 0 -- tone off
  23  72  LDXA     ;Restore D register
  24  70  RET      ;Return -- enable interrupts
  25  C4  NOP      ;Begin -- NOP for sync
  26  22  DEC   R2 ;Decrement stack pointer
  27  78  SAV      ;Push T
  28  22  DEC   R2 ;Decrement stack pointer
  29  52  STR   R2 ;Push D
```

```
022A  19  INC   R9 ;Increment random number "seed"
  2B  E2  SEX   R2 ;NOP for timing
  2C  9B  GHI   RB
  2D  B0  PHI   R0
  2E  F8  LDI      ;Set up R0 to prepare
  2F  00           ;    for DMA action

0230  A0  PLO   R0
  31  98  GHI   R8 ;Check timer in R8.1
  32  32  BZ       ;If already = 0, branch to
  33  38           ;    skip decrementing
  34  AB  PLO   RB ;Decrement timer
  35  2B  DEC   RB ;    by using RB.0 in
  36  8B  GLO   RB ;    oder to preserve DF value
  37  B8  PHI   R8 ;Put new timer value in R8.1
  38  88  GLO   R8 ;Check tone value
  39  32  BZ       ;If = 0, branch to exit
  3A  22           ;    and reset Q (tone off)
  3B  7B           ;Else set Q to sound tone
  3C  28  DEC   R8 ;Tone timer - 1
  3D  30  BR       ;Branch to exit -- do not
  3E  23           ;    reset Q this time
```

ADJUST RB

```
023F  9B  GHI   RB ;Adjust Rb = RB-3 to address
  40  FF           ;    first of 4 display
  41  03           ;    pages
  42  BB           ;Put value in RB.1
  43  D4           ;Return

    ;End of HI-RES MODIFICATIONS
```

THAT SAME OLD LINE

(HI-RES CHIP-8 RANDOM LINE DRAW)

```
Requires:  HI-RES CHIP-8 LANGUAGE @ 0000-0243
       :   BE SURE : 01FE = 1244
       :   AND     : 00DB = correct value for your system

0244  RNDLN :A26E  DOT  --Set I to display bit @ 026E
  46          601F ;Vo=1F--Vo = Vx for display = center X
  48          613F ;V1=3F--V1 = Vy for display = center Y
  4A  R1     :C201 ;RNDOM--V2 = random change of X (Δ X)
  4C          C303 ;RNDOM--V3 = random change of Y (Δ Y)
  4E          C401 ;RNDOM--V4 = random direction (VX) decision

0250          C501 ;RNDOM--V5 = random direction (VY) decision
  52          66FF ;V6=FF--V6 = hex value to compliment  X/  Y
  54          3400 ;SK=0 --If V4=0, do not compliment V2
```

```
0256  *      :8263 ;XOR  --Compliment V2 - reverses VX
  58           3500 ;SK=0 --If V5 = 0, do not compliment V3
  5A  *      :8363 ;XOR  --Compliment V3 - reverses Vy
  5C           C407 ;RNDOM--V4 = Random line length 00-07
  5E           7410 ;V4+10--Add 10 hex for random 10-17 length

0260  R2     :D011 ;SHOW --Show single point of line
  62           8024 ;Vo+V2--Add V2 (ΔX) to Vo (Vx)
  64           8134 ;V1+V3--Add V3 (ΔY) to V1 (Vy)
  66           74FF ;V4-1  --Subtract one from loop count
  68           3400 ;SK=0  --If loop count (V4)=0, skip next
  6A           1260  R2    --Jump to display another point
  6C           124A  R1    --Jump to start another line
  6E  DOT    :8000         --Bit for display
```

*The instruction 8XY3 causes Vx to equal Vx exclusive "ORed" with Vy and is an "undocumented" Chip-8 instruction explained in the August 1978 issue of VIPER, Vol 1, issue 2, pages 2-6.

TINY COMMENTS
By Carmelo Cortes

Since Tiny Basic and Chip-8 share many instructions, I will compare them as the best means of explaination

First T.B. is an integer basic, which means no numbers beyond the right side of the decimal point, which makes it hard to use the majority of already written basic programs. Second the very very slow speed of Tiny Basic, makes game enjoyment very difficult. And the manual looks like it was written at the last minute, what with typos and even missing code here and there. It was supposed to be written for the beginner, but it doesn't explain the functions very well. eg: The Subscripted Variable A(X). (Not knowing basic very well, I had to pick up another manual to understand it).

Here are the commands with my comments.

1)NEW------CLEARS OLD PROGRAM
2)LIST
3)RUN
4)SAVE-----(TO TAPE)
5)LOAD-----(FROM TAPE)
6)CLS------(CLEAR SCREEN)
7)END
8)GOTO
9)GOSUB
10)IF-THEN
11)INPUT
12)LET------(NOT USUALY NEEDED)
13)PRINT
14)REM
15)RETURN---(USED WITH GOSUB)

> 1 to 15 are the regular Tiny Basic commands

16)COLOR- I DON'T HAVE A COLOR BOARD SO I CAN'T COMMENT ON THIS COMMAND

17)GOKEY- HARD TO USE, DOESN'T WAIT, SO YOU MUST EITHER LOOP ON THIS, OR KEEP PRESSING THE KEY WAITING FOR IT TO ACCEPT INPUT

18)PRINT-AT- FANTASTIC! GREAT FOR DISPLAYING SCORES AT SPECIFIC LOCATIONS. DOES THE WORK OF A WHOLE CHIP-8 SUBROUTINE.

19)SHOW- NICE, WORKS EXACTLY LIKE THE CHIP-8 DXYN COMMAND, WILL EVEN REPEAT YOUR PATTERN IN THE X AND/OR Y COORDINATES

21)FQ- FREQUENCY COMMAND, SETS THE FREQUENCY OF THE TONE GENERATOR IN THE SIMPLE SOUND BOARD.

22)TO- TURNS ON THE TONE GENERATOR AND ACCEPTS VALUE FOR TONE DURATION

23)PT- PATTERN` COMMAND, WORKS EXACTLY LIKE THE AMMM CHIP-8 COMMAND

24)TVOFF- USE OF THIS COMMAND WILL ALLOW YOU TO TURN OFF THE DISPLAY, SO PROGRAMS CAN EXECUTE FASTER. (BUT NOT BY MUCH)

25)TVON- TURN DISPLAY BACK ON

26)ABS- THIS COMMAND WILL TURN NEGATIVE NUMBERS INTO POSITIVE

27)RND- TERRIBLE, THIS COMMAND IS NON-MASKABLE, YOU GET A RANDOM NUMBER FROM Ø TO 255 AND THATS IT. IF YOU WANT A LOWER VALUE, YOU MUST PUT IN AN IF-THEN STATEMENT TO LOOP BACK IF THE VALUE IS TOO HIGH, AND THE WAIT IS INTOLORIABLY LONG

28)HIT- FUNCTIONS THE SAME AS THE CHIP-8 VF VARIABLE, IF A HIT HAS OCCURED "HIT" IS SET TO 1

29)KEY- THIS CONTAINS THE ASCII VALUE OF THE KEY THAT WAS PRESSED DURING GOKEY.

30)MEM- THIS COMMAND TELLS YOU HOW MUCH MEMORY IS LEFT

As you can see some of the special Tiny Basic commands, 16 to 30, duplicate some of the Chip-8 commands almost exactly. In fact I was able to rewrite one of my UFO games into Tiny Basic almost instruction for instruction. The only instruction that saved space, was the Print At command. This command took the place of a whole Chip-8 subroutine. (again great for displaying scores) When I ran the program, I found that I had wasted my time. The game ran so slowly, that the spaceship nearly took five minutes to move across the screen. This gave Chip-8 a whole new look to me, trying to use the graphics function in Tiny Basic, is almost a total waste of time.

Another thing that didn't sit well with me, was that you can't run Chip-8 or Machine Lang. programs with the T.B. Board plugged in, you have to remove the board, then load your code as usual.

I can go on and on, but I don't want to get carried away. Maybe I expect too much from Tiny Basic. Either way my opinion is that, Chip-8 is superior in many ways, and that this one fellow Viper, that will continue to use Chip-8 and praise it.

Carmelo Cortes
73 Catherine St.
Hartford, CT. 06106

8 SECTOR, 2 PAGE KALEIDOSCOPE

by George Ziniewicz

I like to explore the graphics capabilities of the VIP, and this article describes one of the best examples of graphic manipulation I've come up with so far.

Basically, the program takes a set of X,Y coordinates and generates a mirror image set. Then, using the two sets of coordinates, it displays four dots on the screen. Sound similar to standard VIP Kaleidoscope?

The difference lies in the fact that in two page displays, the X and Y resolution are equal (that is, X and Y have the same upper limit). Thus a DXYN instruction can be followed by a DYXN instruction. If X is not equal to Y (no real problem), you can obtain twice the number of sets of X,Y coordinates as is generated by the standard version. The DYXN operation yields eight sets of X,Y positions (compared to the four sets obtained from the standard Kaleidoscope), providing a very pleasing symmety on the screen.

The program uses a subroutine to alter the X,Y coordinates, allowing you to easily change the algorithm used. Freeze (F) and Continue (C) are present in this program, along with a provision for randomly erasing the screen to avoid congestion. The Erase can be bypassed, of course, for greater filling in.

I've shown a collection of X,Y move algorithms I've found to be visually stimulating, but feel free to experiment.

The program replaces the wait for interrupt instruction in the display routine with a nop, for speed. (Replace 00 at location 00AC with EC.) This is a useful substitution in almost all programs, yeilding an average of 300% increase in the speed. You can get almost instantaneous full screen displays of information! Try it in Wipe Off, Bowling, etc. If you like, try this Kaleidoscope program without making the substitution, and notice the difference in the speed.

ADDR	CODE	COMMENTS
0300	A3F0	Dot map
0302	23A0	Do X,Y Control
0304	6CFF	Make mirror image
0306	6DFF	..of A,B in C,D
0308	8CA5	
030A	8DB5	
030C	DAB1	DXYN
030E	DAD1	
0310	DCB1	
0312	DCD1	
0314	DBA1	DYXN
0316	DDA1	
0318	DBC1	
031A	DDC1	

```
031C    CEFF    Erase Display
031E    4EED    ..every 4 sec.
0320    0230    ..randomly
0322    6F0F    Continue if
0324    EF9E    ...key F has not
0326    132E    ...been pressed
0328    6F0C    Wait for Key C
032A    EF9E    ...to be pressed
032C    132A    ..before continuing
032E    1302    Repeat; or, user can include other functions
                beginning here.
```

8 SECTOR 2 PAGE KALEIDOSCOPE X,Y MOVE SUBROUTINES TO TRY OUT

```
03A0    7A01
        7B02    (or 7B03)
        00EE

03A0    CF01
        4F00
        7A01
        4F01
        7B01    (or 7B02)
        00EE

03A0    CAFF
        CBFF
        00EE

03A0    CAFF
        7B01    (or 7B02)
        00EE

03A0    CF01    (or CF03 or
        4F00    ..CF07, etc.)
        7A01
        7B01
        00EE

03A0    CC03
        CD03
        7CFE
        7DFE
        8AC4
        8BD4
        00EE
```

VIP VOX

by George Ziniewicz

The "Voice Operated Switch" function is easy to implement on the VIP with this MLS, which scans the cassette input line for a specific time interval. It's looking for a signal of minimum duration, setting VF to 0 if no signal is found or to 1 if a signal is present and long enough to be picked up by the scan.

Input to the VIP can be from an amplified microphone (voice) or from radio, tape, etc. (music). A simple color organ can be set up by inserting a call to the VOX in the middle of the Kaleidoscope program loop, or in any other graphic program, and looping until VF=1.

```
0XXX    Branch to VOX
4F00    If no sound,
1       ...loop back
```

Insert in a program...it will wait until a sound is input

0XXX	Branch to VOX	In games with Fire buttons,
4F01	If sound present,	it will fire weapon upon
2	Branch to Fire sub.	voice command instead of keypress.

VOX has been expanded to indicate duration of input signal in 30 microsecond units in VF, to allow for simulating a graphic frequency meter (to be described in a later issue).

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0X00	F8	20	AC	F8	FF	A6	F8	00	56	00	3D	11	2C	8C	3A	0A
0X10	D4	F8	06	AC	35	1D	2C	8C	3A	14	F8	01	56	D4		

The scan time (set in byte 01) =n*40 microseconds, or 1.28 ms. The scan time is placed in RC.

Bytes 03-05 point to VF, and bytes 06-08 set VF=0. Byte 09 waits for a signal, and if one is present, bytes 0A and 0B branch to 0X11. Bytes 0C through 0F check for a signal until the time is up, looping back to byte 0A. Byte 10 returns to the calling program if no signal has been found during the specified time limit, or upon return from the subroutine at byte 11.

Bytes 11 through 13 store the signal duration in RC. The minimum duration of signal = n*40μs, or 240μs, 2KHz signal. At byte 14, if there has been no signal, the subroutine is exited. Bytes 15 through 19 continue to check until the time is up. If a signal has been found, bytes 1A through 1C set VF to 1, and then byte 1D returns to the calling subroutine (at byte 0C).

Adapted from an idea I got from a BOZO TV show, I incorporated VIP VOX into a simple program that fires a rocket at a space-ship upon voice command, and called it POW. My 18-month old son is fascinated - and fantastic at it!

MODIFICATIONS TO PIPS FOR VIPS PROGRAM EDITOR

by Steven Medwin

Tom Swan's Chip-8 program editor, as published in PIPS FOR VIPS Vol 1, is an excellent program. It has some distinct advantages over the Hersch editor, most notably its speed, but lacks some of Hersch's best features. I will describe a series of modifications to the Swan editor that add new capabilities to make it a very powerful software tool.

First of all, I wanted to relocate the editor so it could be used along with the Chip-8 interpreter. This would permit easy development of Chip-8 programs by allowing execution of the program, just as with the Hersch editor. Along with this, a command to execute Chip-8 programs was needed. Secondly, I wanted to add the address command found in the Hersch editor. This facilitates jumping around a program, instead of scrolling to an address. These modifications take up a little more room than the original editor, but it still fits in only two pages of memory.

Once I made these modifications, I realized I could use the editor for more than just software development. One example is the Kaleidoscope program. By using the editor with this program, the sequence of keystrokes used to create a pattern can be easily displayed after the pattern has been run. I find this useful when I discover a pattern I like and want to record the sequence.

Another use for the editor is with Pin-8 and the Supersound board. The editor makes it much easier to enter or modify a song than with the VIP's operating system. Then by pressing only one key, the new song can be heard. As a matter of fact, any machine language program can be developed and tested using the editor. (I have used these two applications, Kaleidoscope and Pin-8, as examples of how to combine the editor with a finished program. These examples can be found at the end of this article.)

The modifications that have to be made can be broken down into two parts. The first part are modifications necessary to relocate the editor. The second part are the instructions needed to implement the extra commands. A listing of these changes can be found at the end of this article. One mote about making these changes: It's easiest to use the unmodified editor to modify a copy located in high memory.

The new location for the editor was set by the requirements of the Chip-8 interpreter used in PIPS FOR VIPS. This version used the two page display, the messager routine and had the Chip-8 work area located in page 2. Consequently the editor was located in pages C and D of my 4K VIP, with pages E and F reserved for the display. A memory map showing this organization can be found on the next page.

Memory Map

Page	Use
0 1	Chip-8 interpreter
2	Two page display, messager routines and Chip-8 work area
3 : B	Chip-8 program (9 pages available)
C D	Modified editor
E F	Two page display

A standard Chip-8 interpreter with the one page display can also be used. The only change is that the work area must be moved from page 2 to page E (set M(ØDD5) to "ØE").

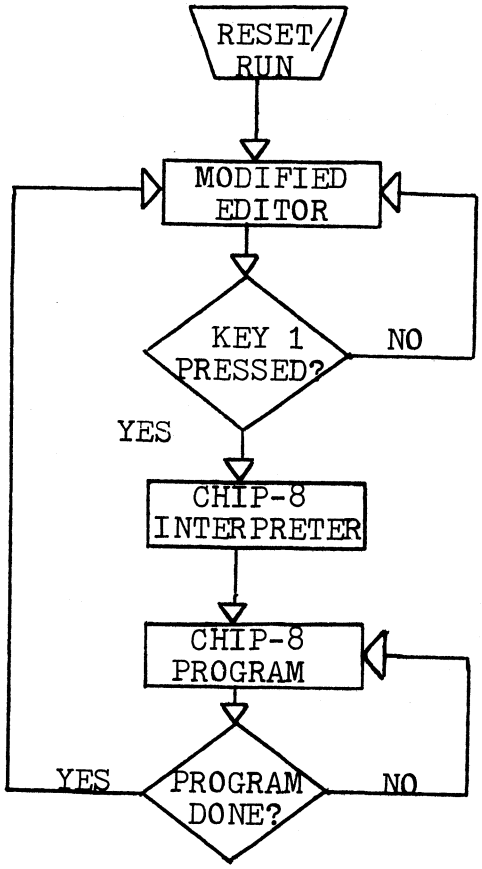
The operation of the modified editor might seem a bit confusing, so I included a flow chart showing how it works. When the run switch is flipped up control jumps immediately to the editor. All editor commands can be used until key 1 is pressed: then control is passed to the Chip-8 interpreter. Any Chip-8 program is then executed as normal. When this program ends a simple call to a machine language subroutine, located in the editor, can be used to pass control back to the editor. The examples at the end of this article should be studied to understand how to implement this in your application.

Once you've made all these changes, you may be wondering how to use the new features. All of the original commands are implemented just as described in PIPS FOR VIPS, except that the scrolling speed has been increased (the byte at M(ØCF6) controls this - see listing). The address command is simple: press key A followed by the three(3) digit address (the first zero is not entered). The editor will then show this address as the last one in the display. The execute command is even simpler - just press key 1 and your Chip-8 program will start running.

The modified editor will make your VIP much easier to use. You can use it for program development or incorporate it in a finished program (as I did with Kaleidoscope and Pin-8). Or, if you're so inclined, you can modify it to make it even more useful (and then write an article about it for The VIPER).

(continued next page)

Flowchart of Operation with Modified Editor



Listing of Modifications to Relocate Editor:

Jump to Editor

```
0000 90
     A3          00 ⟶ R3.0
     F8 0C       0C ⟶ R3.1
     B3          "0C" is first page of editor
  05 D3          Program counter    R3
```

Micellaneous Instructions

```
0C07 E2     NOP
  13 0D     Second page of editor
  20 03     Initial display address - high byte
  2F 00        "       "         "    - low byte
  30 A4
  D7 0D     Second page
  DE 0C     First page
  F6 06     Sets scrolling speed
0D41 0D     Second page
  AA 0C     First page
```

Add to Extra Commands:

```
0C80-0C8E     Jump to new commands
0D33-0D3F     Jump from Chip-8 to editor
0DB0-0DCD     Enter new address  Key A
0DCE-0DE0     Execute program    Key 1
0DE1-0DF6     Page backward      Key D
0DF8-0DFF     Reserved for editor stack
```

Jump to New Commands

```
0C80 F8 0D
     BF
     F8 B0    Jump to 0DB0  Program counter ⟶ RF
     AF
     DF
0C87 F8 0C
     B3
     F8 8F    Jump to 0C8F  Program counter ⟶ R3
     A3
     D3
0C8E 00
```

Jump from Chip-8 to Editor (call this machine language subroutine from Chip-8 with "0D33" command)

```
0D33 61       Turn display off
     E0       0 ⟶ X
     F8 0F
     B1
     F8 FF    0FFF ⟶ R1
     A1
     F8 00
     A0       0000 ⟶ R0
     B0
0D3F D0       Program counter ⟶ R0
```

Enter New Address

```
0DB0 F8 0D
     B3
     F8 B7    0DB7 ⟶ R3
     A3
     D3       Program counter ⟶ R3
  B7 F8 0A
     F3
     3A CE    Key = A?
     DC
     F3
     3A CE    Key debounce
  C0 DC       Enter 1st digit
     B4       Store in R4.1
     DC       Enter 2nd digit
     FE FE FE FE   Shift left 4 times
     A4       Store in R4.0
     DC       Enter 3rd digit
     84
     F4       Add 3rd digit to R4.0
     A4       Store result in R4.0
  CC 30 A9    Branch to 0DA9 for return to main routine
```

Execute Program

```
0DCE F8 01 )
     F3    }   Key = 1?
     3A E1 )
     DC          Key debounce
  D4 F8 02 )
     B2    }     Part of Chip-8 initialization-sets page for
     B6 )           work area
  D8 E2 E2       NOP
     F8 00 )
     BF    }     0006⟶RF
     F8 06 )
     AF
  E0 DF          Program counter⟶RF (branch to Chip-8
                                     interpreter)
```

Page Backward

```
0DE1 F8 0D )
     F3    }   Key=D?
     3A F0 )
     F8 0C       0C⟶RF
     AF
  E9 24 )
     2F    }     Loop to decrement address pointer
     8F    }
     3A E9 )
     30 A9       Branch for return to main routine
  F0 F8 0C )
     BF    }     0C87⟶RF
     F8 87 )
     AF
  F6 DF          Program counter⟶RF
```

Example 1 - Using the editor with Kaleidoscope the following steps will modify the Kaleidoscope program as described in the article:

1. Load the modified editor
2. Load Kaleidoscope starting at 0000. (Use the version by Phil Sumner in VIPER 1.09.13.)
3. Change 0000-0005 as listed earlier.
4. Make following additional changes:

```
0358  F90A       Key debounce
  5A  0D33       Maching language call that jumps to editor
0C20  02    }    Change display address to 0280
  2F  80    }
0DD5  0E         Set Chip-8 work area to P6.E.
```

5. Record all this on tape (E pages)
6. Flip run switch up - should see addresses 280-288.
7. Press key 1 to run Kaleidoscope as usual.
8. Press key A to display keystroke sequence (if screen blanks out, press key A again).

Example 2 - Using the editor with Pin-8 (for Supersound board) the following steps will modify Pin-8 as described in the article:

1. Load the modified editor.
2. Load Pin-8 with a song starting at 0000.
3. Change 0000-0005 as listed earlier.

4. Make following additional changes:

```
007A  F2          Change branch address
00E0  61          Turn off display
      E0
      F8 0F
      B1 B2 BE    Set up for Pin-8 initialization
      F8 FF
      A1 A2
00EB  F8 00
      B0
      F8 07       0007 ---> R0
      A0
00F1  D0          Program counter ---> R0
00F2  E0
      F8 0F
      B1          Set-up for editor initialization
      F8 FF
      A1
      F8 00
      A0 B0       0000 ---> R0
  FD  D0          Program counter ---> R0
0C20  02
  2F  70          Display address
0DDE  E0          Branch address
```

5. Set last break control in song to "FF" (to stop).
6. Record this on tape (E pages).
7. Flip run switch up - should see addresses 270-278.
8. Press key 1 to play song.
9. When song ends, control will jump to editor.
10. Modifications can now be made and then the song replayed by pressing key 1.

(We received a letter from C Spencer Powell, in which he pointed out to Tom that several interesting and helpful items of info were omitted from PIPS I. To atone for the omissions, Tom has responded to Mr. Powell here - so the rest of us can benefit from his reply. - Terry)

Dear Mr. Powell,

I'm sorry the cassette gave you problems. Perhaps a friend could loan you a different brand of tape player to see if the tape will load. That sure is a lot of code to key in by hand!

Everything you need to complete your programs is listed below. The character set was not included as the Character Designer program was written so that you would be able to create your own characters. If you would rather not do this, the entire set is reproduced here. The same code is used by Text Editor-21, Disassembler-7 and Character Designer. (As a bonus, I've included an improved lower case set here which other Pips owners may want to examine.)

ASCII CHARACTER SET

```
0M00 - 0M7F - Not important - control characters 00-1F
0M80  00 00 00 00 22 20 20 00 55 00 00 00 17 27 40 00
0M90  27 43 70 00 66 44 32 30 20 70 20 00 24 00 00 00
0MA0  12 22 21 00 42 22 24 00 05 25 00 00 02 72 00 00
0MB0  00 00 24 00 00 30 00 00 00 00 40 00 10 20 40 00
0MC0  75 55 70 00 26 22 70 00 71 74 70 00 71 31 70 00
0MD0  45 71 10 00 74 73 70 00 74 75 70 00 77 11 10 00
0ME0  75 75 70 00 75 71 70 00 00 20 20 00 00 10 12 00
0MF0  12 42 10 00 07 07 00 00 42 12 40 00 71 20 20 00

0N00  07 54 70 00 77 57 50 00 65 75 60 00 76 66 70 00
0N10  73 33 70 00 76 76 70 00 76 76 60 00 74 57 70 00
0N20  55 75 50 00 72 27 70 00 11 57 70 00 45 66 50 00
0N30  44 47 70 00 57 75 50 00 47 77 50 00 77 55 70 00
0N40  77 57 40 00 77 55 60 00 75 76 50 00 76 73 70 00
0N50  77 72 20 00 55 57 70 00 55 55 20 00 55 77 50 00
0N60  55 25 50 00 55 22 20 00 71 24 70 00 32 22 23 00
0N70  40 20 10 00 62 22 26 00 25 00 00 00 00 00 00 70
0N80  21 00 00 00 02 57 50 00 44 75 70 00 00 76 70 00
0N90  11 75 70 00 00 75 67 00 32 72 20 00 00 75 71 70

0NA0  44 75 50 00 02 02 20 00 02 02 26 00 44 56 50 00
0NB0  02 22 20 00 00 57 50 00 00 67 50 00 00 75 70 00
0NC0  00 75 74 40 00 75 71 10 00 76 60 00 07 61 70 00
0ND0  02 72 20 00 00 55 70 00 00 55 20 00 00 57 50 00
0NE0  00 52 50 00 00 55 71 70 07 12 70 00 32 42 30 00
0NF0  22 02 20 00 62 12 60 00 00 63 00 00 00 00 00 00
```

Space Wars uses a slightly modified character set to display the title. These modifications go in <u>after</u> loading the full character set above into 0700-08FF.

```
0880  8C DD F7 30 26 66 EC 80 37 EC FC 80 8C E6 E6 20
0890  FF CC FC 80 8C CC 8C 60 7F C7 07 F0 CE 0C 6E C0
08A0  78 BA B8 70 84 44 44 80 01 37 F0 00 EB 5B F0 00
08B0  00 8C E0 00
```

The Mnemonic Lookup Table and Argument lookup table you were looking for in Disassembler-7 were left for the reader to construct. This was done to allow you to modify it to your own tastes, but I agree with you, leaving it out wasn't such a good idea afterall. Sorry. Here it is.

```
0800  00 49 44 4C 00 0F 4C 44 4E 00 1F 49 4E 43 00 2F
0810  44 45 43 00 30 42 52 00 31 42 51 00 32 42 5A 00
0820  33 42 44 46 00 34 42 31 00 35 42 32 00 36 42 33
0830  00 37 42 34 00 38 53 4B 50 00 39 42 4E 51 00 3A
0840  42 4E 5A 00 3B 42 4E 46 00 3C 42 4E 31 00 3D 42
0850  4E 32 00 3E 42 4E 33 00 3F 42 4E 34 00 4F 4C 44
0860  41 00 5F 53 54 52 00 60 49 52 58 00 6F 4F 55 54
0870  00 61 49 4E 50 00 70 52 45 54 00 71 44 49 53 00
0880  72 4C 44 58 41 00 73 53 54 58 44 00 74 41 44 43
0890  00 75 53 44 42 00 76 53 48 52 43 00 77 53 4D 42
```

```
08A0  00 78 53 41 56 00 79 4D 41 52 4B 00 7A 52 45 51
08B0  00 7B 53 45 51 00 7C 41 44 43 49 00 7D 53 44 42
08C0  49 00 7E 53 48 4C 43 00 7F 53 4D 42 49 00 8F 47
08D0  4C 4F 00 9F 47 48 49 00 AF 50 4C 4F 00 BF 50 48
08E0  49 00 C0 4C 42 52 00 C1 4C 42 51 00 C2 4C 42 5A
08F0  00 C3 4C 42 44 46 00 C4 4E 4F 50 00 C5 4C 53 4E
0900  51 00 C6 4C 53 4E 5A 00 C7 4C 53 4E 46 00 C8 4C
0910  53 4B 50 00 C9 4C 42 4E 51 00 CA 4C 42 4E 5A 00
0920  CB 4C 42 4E 46 00 CC 4C 53 49 45 00 CD 4C 53 51
0930  00 CE 4C 53 5A 00 CF 4C 53 44 46 00 DF 53 45 50

0940  00 EF 53 45 58 00 F0 4C 44 58 00 F1 4F 52 00 F2
0950  41 4E 44 00 F3 58 4F 52 00 F4 41 44 44 00 F5 53
0960  44 00 F6 53 48 52 00 F7 53 4D 00 F8 4C 44 49 00
0970  F9 4F 52 49 00 FA 41 4E 49 00 FB 58 52 49 00 FC
0980  41 44 49 00 FD 53 44 49 00 FE 53 48 4C 00 FF 53
0990  4D 49 00 3F 3F 3F 00 00

09C0  F8 F9 FB FA FC 7C FD 7D FF 7F 30 31 32 33 34 35
09D0  36 37 39 3A 3B 3C 3D 3E 3F 00 C0 C2 CA C3 CB C1
09E0  C9 00 00 00 00 00 00 00
```

Finally, you were looking for the ASCII coded messages for Space Wars, at 06A4-06FE on page 119. Actually nothing at all should go in those memory locations -- they are supposed to be blank. All ASCII messages were included with the listing in Pips.

That should do it. I appreciate your taking the time to write. Please feel free to write back and let us know what you think of the programs when you get them running.

Regards,

TOM

Tom Swan

A FIX FOR PIPS II

Many readers have written to tell us of problems with the up/down key (among other things) they have with the PIPS II programs. The problem was finally traced to the fact that when Tom wrote PIPS (all three volumes) he had a prototype keyboard to work with. When RCA finally released the VP601, they had incorporated some design changes - which means you get to do some re-wiring if you want to run PIPS II.

Cut the data line 7 coming from the keyboard, and leave the keyboard end of the cut line unterminated. Ground the VIP end of the line - and you should be all set. - Rick & Terry

LITTLE LOOPS by Tom Swan

THE SORTED DETAILS

One of these days you will run into a situation where you will want to put some things in order that happen to be very much out of order. Now no algorithm in the world is going to force order onto the streets near us that twist and turn through the Mexican market with its confusion of stalls and shops and Indian women selling dried beans from their laps. To sort such a thing would be to remove its charm. I was referring, of course, to things such as numbers in a computer program, though there are those who would modernize that beautiful old world colonial maze and thereby destroy it.

Sorting numbers is one of the focal points of the computer sciences. Donald Knuth covers about 25 various modern sorting algorithms in Volume 3 of the Art of Computer Programming. At first it may seem to be a silly subject to get all worked up about, but let's say you have been recording the scores of a series of Chip-8 games and you want to output an orderly list of these scores from the lowest to the highest. Just how do you go about arranging that list?

BURSTING A BUBBLE

A lot of popular computer literature speaks of a sorting algorithm called the Bubble Sort. It has become one of those "buzz words" that writers use when they want to suggest a possesion of knowledge they may or may not have the grip on they want you to believe. Let's Burst that Bubble, so to speak, and demonstrate how the Chip-8 language may be used to experiment with computer concepts such as searching and sorting.

With your normal Chip-8 interpreter loaded in locations 0000-01FF, enter the following program, but do not run it yet. Save three pages on tape from 0000 and mark it as the Sort Test #1.

CHIP-8 -- SORT TEST #1

```
0200  TEST1 ;6100 ;V1=00 -- Initialize display variables
  02        6200 ;V2=00 --    "         "        "V1=VX/V2=VY
  04        6D00 ;VD=00 -- Initialize count to 00 (VD=# inputs)
  06  T1    ;A400  RECRD -- Set "I" to 0400 where values will go
  08        FD1E ;I+VD  -- Index "I" to next storage space
  0A        F00A ;V0=KY -- Wait for input, then set V0 = key
                           pressed value
  0C        F055 ;PUT   -- Store V0 @ I+VD
  0E        7D01 ;VD+1  -- Count the input
```

```
0210          F029 ;BITS  -- Let I = bit pattern for V0
  12          D125 ;SHOW  -- Display input value in V0
  14          7105 ;V1+05 -- VX+05 for next input display
  16          3000 ;SK=00 -- Skip next if V0=00 (Key 0
                             calls sort)
  18          1206  T1    -- Jump to get another keypress
  1A          2300  SORT  -- Do sub -- Sort the Values
  1C          6004 ;V0=04 -- V0 = value for tone
  1E          F018 ;TONE  -- Signal when done with sort

0220          6100 ;V1=00 -- V1 = VX for display
  22          6208 ;V2=08 -- V2 = VY for display
  24          6C00 ;VC=00 -- VC = index to sorted results
  26  T2     :A400  RECRD -- Set "I" to address of records
  28          FC1E ;I+VC  -- Index to next record
  2A          F065 ;GET   -- Let V0=M(I)=the record @ I+VC
  2C          F029 ;BITS  -- Let I = bit pattern for V0
  2E          D125 ;SHOW  -- Display the value at V1 V2

0230          7105 ;V1+05 -- Add 5 to VX for next digit
  32          7C01 ;VC+01 -- Add 1 to index for next value
  34          5CD0 ;SK=   -- If index VC = number records
                             VD, skip
  36          1226  T2    -- Jump to display another record
  38  HERE   :1238  HERE  -- Stop
```

This routine allows you to enter a sequence of numbers ($N \leq 13$) ending with the value "0" which calls the sorting sub. The values are stored sequentially at 0400 to 0400+VD where VD = the number of entries.

Now we are ready to develop our first sorting routine, a Bubble Sort straight from an algorithm in Donald Knuth's book.

A Bubble Sort works by exchanging pairs if the first in the pair is greater than the second. All the values are tested in pairs from left to right until the limit of the set is reached. When such a particular pass results in no exchanges, the set is sorted. The upper limit of the set is changed to the position of the last value to be exchanged. Since the set's highest values "bubble up" to the top during each pass, the last value to be swapped never needs to be considered again and the process of setting the upper limit can be used to flag the occurrence of an exchange for the sake of efficiency.

Bubble Sort Example

5;3;1;2
3;1;2;5
1;2;3;5

The following subroutine sorts a list of numbers according to the Bubble Sort algorithm. You may enter it at 0300, then save it on tape either separately or along with the Sort Test #1 you entered earlier. Run the program several times to sort a series of numbers ($N \leq 13$) which you enter on the hexpad of your computer.

CHIP-8 BUBBLE SORT

```
0300  SORT  :8AD0 ;VA=VD -- Set # records into VA (+)
  02  S1    :8CA0 ;VC=VA -- Set VA into VC (= upper limit)
  04          6B01 ;VB=01 -- VB indexes data--initialize VB=1
  06          6A00 ;VA=00 -- Flag=0 - VA also establishes limit
  08          1326  LIMIT -- Jump to limit to begin sort @ I+VB
  0A  S2    :A3FF RECRD-1-- Set "I" = address record -1
  0C          FB1E ;I+VB  -- Index "I" to record (VB)
  0E          F165 ;GET   -- Let V0=record VB/V1=record VB+1

0310          8210 ;V2=V1 -- Save V1 (R(VB+1)0 in temporary
                             variable V2
  12          8105 ;V1-V0 -- Subtract R(VB+1)) - R(VB)
  14          3F00 ;SK=00 -- If VF=0, skip (negative result--
                             (R(VB)> R(VB+1))
  16          1324  S3    -- Jump to skip the exchange next
  18          8100 ;V1=V0 -- Transfer R(VB) into V1 (swap)
  1A          8020 ;V0=V2 -- Transfer R(VB+1) into V0 (swap)
  1C          A3FF RECRD-1-- Set "I" = address record -1
  1E          FB1E ;I+VB  -- Index "I" to record (VB)

0320          F155 ;PUT   -- Store swapped reocrds held in V0,V1
  22          8AB0 ;VA=VB -- Set VB into VA to flag exchange
                             and set limit
  24  S3    :7B01 ;VB+01 -- Increase VB to get next record
  26  LIMIT :5BC0 ;SK=   -- If VB=VC then upper limit reached,
                             skip
  28          130A  S2    -- Jump to continue sorting
  2A          3A00 ;SK=00 -- If VA=00 then there were no
                             swaps, exit
  2C          1302  S1    -- Jump to continue till no swaps
                             occur
  2E          00EE ;RET   -- Return--exit sub-sort complete
```

The second sorting algorithm is of a completely different nature than a Bubble or an exchange algorithm. Again we will use it along with the Sort Test program that you entered earlier.

This technique is called an "Insertion Sort" and while it does not enjoy the popularity of its distant Bubble-headed cousin, it wins hands down in a competition as we will dramatically show a little later.

The Insertion Sort may be thought of as a "sinking sort" since values tend to move downwards in the list arriving, eventually, at their proper location. The process resembles the way you may arrange a set of file cards alphabetically in a box. You flip through the cards in the box moving each aside until you come to the probable place for the card in your hand, the one you wish to insert. You are making the assumption that all the other cards in the box are already in order though even if this is not thru, the process works when we repeat it for each card in the box.

The insertion sort begins by comparing the second number with its neighbor to the left. If that neighbor is greater than the number in question, the neighbor is moved up one to the right. When a neighbor is less than or equal to the number, that number is inserted to the right of the neighbor. (Consider position #0=0) Each value to the right of the second value is examined with its neighbors to the left until all values have been checked.

Insertion Sort Example
5;3;1;2
3;5;1;2
1;3;5;2
1;2;3;5

CHIP-8 INSERTION SORT

```
0300  SORT  :6001 ;V0=01 -- Set a utility variable = 01
  02         80D5 ;V0-VD -- Subtract # records in VD from V0
  04         3F00 ;SK=0  -- If VF=0, then result negative--
                            VD≥2
  06         00EE ;RET   -- Return if there are less than 2
                            records
  08         7D01 ;VD+01 -- Add 1 to VD for later test
  0A         6B02 ;VB=02 -- Begin with 2nd record--VB is index
  0C  S1    :A3FF ;RECRD-1- I= record address -1
  0E         FB1E ;I+VB  -- Index I to record (VB)

0310         F065 ;GET   -- V0=Record (VB)
  12         8C00 ;VC=V0 -- Place record in VC (place in hand
                            -- see text)
  14         8AB0 ;VA=VB -- VA will index values to the left
                            for comparing
  16  S2    :7AFF ;VA-1   -- Start with record (VB-1)
                            (VA=VA-1 here on loops)
  18         A3FF ;RECRD-1- I=record address -1
  1A         FA1E ;I+VA  -- Index to record (VA)
  1C         F065 ;GET   -- V0= record (VA)  (I=I+1)
  1E         81C0 ;V1=VC -- To preserve VC, place it in V1

0320         8105 ;V1-V0 -- Subtract V1-V0 (record (VB) -
                            record (VA))
  22         3F00 ;SK=0  -- If VF=0, then result negative,
                            skip
  24         132C  S3    -- Jump if M(VB)≥M(VA)
  26         F055 ;PUT   -- Put record (VA) in M(VA)+1 to
                            move up
  28         3A00 ;SK=0  -- If VA=0, skip into next part
  2A         1316  S2    -- Jump to continue
  2C  S3    :A400 ;RECRD -- I= record address
  2E         FA1E ;I+VA  -- Index to I(VA)+1

0330         80C0 ;V0=VC -- V0 will transfer current record
                            to memory
  32         F055 ;PUT   -- Store V0 @ M(I)
```

```
0334        7B01 ;VB+01 -- Add 1 to VB
  36        5BD0 ;SK=   -- If VB=VD then skip
                           (VD=VD+1 here)
  38        130C ;S1    -- Jump to continue
  3A        7DFF ;VD-1  -- Reset VD
  3C        00EE ;RET   -- Return
```

THE TORTOISE AND THE HARE

Ok, let's have a race. First of all we need to create a set of random data to sort -- all single byte values, say 100 of them to make things interesting. Unplug your computer then plug it back in. You have just created a set of random data in memory. We can not really assume anything about that data's true randomness but it will do for our purposes.

Of course, you also just wiped out your program. (You did save it on tape as I suggested, I hope?) Below is a simple routine that will allow you to view the actual memory bytes while they are being sorted by simply displaying that memory area on your video screen. You need to have your Chip-8 interpreter in location 0000-01FF. Then enter one of the two sorting routines listed here, preferably the Bubble Sort first. Enter the following Sort Test #2 (you do not need to record it, but you may) and flip the run switch up. Repeat for the Insertion Sort by first unplugging your computer, entering the interpreter and sort sub then the Sort Test #2 again and running.

Surprised at the results? Both are pretty slow aren't they? My 100 bytes took 46 seconds with the Bubble Sort and 16 seconds with the Insertion technique! That's about 65% faster for the Insertion variety, a remarkable improvement over the often-quoted Bubbler.

SORT TEST #2

```
0200  TEST2 :0208  MLS    -- Change display page to 0400
  02         6D64 ;VD=64  -- Set # elements=64 hex = 100
  04         2300  SORT   -- Call sort
  06  HERE   :1206  HERE   -- Stop
  08  MLS    :F804         -- Machine language subroutine to
  0A         BBD4          --    change display page
```

Remember that you are viewing your data upside down (relative to the construction of the subs and the video circuitry) so that the Bubble Sort appears to actually Bubble down -- which is not a new betting strategy in Blackjack. (If you don't get it, please excuse me and go on.)

Notice the number of exchanges occurring with a Bubble Sort. Then compare with the action of the Insertion Sort watching

how data is simply moved aside until the assumed location for the byte in question is discovered. You are seeing something which would not be possible to view on many computers -- the actual memory bytes in action.

Now that you've got everything in order, we will discuss how to scramble everything up again -- one of these months. (Computer programmers are never satisfied!) Hasta la vista.

LAST MONTH'S ANSWER:

I overreached my own memory Chips a bit last month -- I don't really know who wrote the song Short Shorts. However, I do know that some of the same fellows went on to form a group called The Four Seasons which were something of a splash at one time. End of non-computer trivia corner.

PROJECTS: (answers next month)

1) I purposely neglected discussing what happens to equal values during a sort. What does happen? Do equal values retain their relative order following each of the sorts mentioned?

2) Sort the following sequence on paper using both algorithms writing down each step until the sequence is arranged in order: 9;1;3;7;2;6;4;8;5

3) The Sort Test #1 waits until all data has been entered to begin sorting. Write a test that sorts each value as it is entered. Which of the two sorts lends itself to this type of data handling?

4) Devise a better test of the relative speed of both sorting subs using the random number generator to produce the data. Have the computer keep track of the relative times required. Run several tests, then sort the results and output a list. (For more advanced study -- answer to this not to be given next month.)

NEXT MONTH: IT'S GETTING BETTER ALL THE TIME